QUESTION 2:

Multiple choice question **[6 Marks]**

1. What is the name of the method used to start a thread execution?
   a. init();
   b. start();
   c. run();
   d. resume();

2. Which of the following will not directly cause a thread to stop?
   a. notify()
   b. wait()
   c. InputStream access
   d. sleep()

3. Which of the following will directly stop the execution of a Thread?
   a. wait()
   b. notify()
   c. notifyall()
   d. exits synchronized code

4. Which three guarantee that a thread will leave the running state?
   a. yield()
   b. wait()
   c. notify()
   d. notifyAll()
   e. sleep(1000)
   f. aLiveThread.join()
   g. Thread.killThread()

5. class X implements Runnable
   {
   public static void main(String args[])
   {/
   * Missing code? */
   }
   public void run() {}
   }
   Which of the following line of code is suitable to start a thread?
   a. Thread t = new Thread(X);
   b. Thread t = new Thread(X); t.start();
   c. X run = new X(); Thread t = new Thread(run); t.start();
   d. Thread t = new Thread(); x.run();

6. Assume the following method is properly synchronized and called from a thread A on an object B:
   wait(2000);
   After calling this method, when will the thread A become a candidate to get another turn at the CPU?
   a. After thread A is notified, or after two seconds.
   b. After the lock on B is released, or after two seconds.
   c. Two seconds after thread A is notified.
   d. Two seconds after lock B is released.

7. Which cannot directly cause a thread to stop executing?
   a. Calling the SetPriority() method on a Thread object.
   b. Calling the wait() method on an object.
   c. Calling notify() method on an object.
   d. Calling read() method on an InputStream object.

8. public class MyRunnable implements Runnable

```
{
    public void run()
    {
    // some code here
    }
}
```

Which of these will create and start this thread?

a. new Runnable(MyRunnable).start();
b. new Thread(MyRunnable).run();
c. new Thread(new MyRunnable()).start();
d. new MyRunnable().start();

9. Which method must be defined by a class implementing the java.lang.Runnableinterface?

a. void run()
b. public void run()
c. public void start()
d. void run(int priority)

10. Which will contain the body of the thread?

a. run();
b. start();
c. stop();
d. main();

11. Can we make the user thread as daemon thread if thread is started?

a. Yes
b. No

12. You designate a method as being synchronized by:

a. Using the synchronized keyword
b. Placing a synchronized section of code within it
c. Naming the method synchronize

Q2. What will be the output of the program? [4 Marks]

```
//RunnableExample.java
class Count implements Runnable
{
   Thread mythread ;
   Count()
   {
     mythread = new Thread(this, "my runnable thread");
     System.out.println("my thread created" + mythread);
     mythread.start();
   }
   public void run()
   {
     try
     {
       for (int i=0 ;i<5;i++)
       {
         System.out.println("Printing the count " + i);
         Thread.sleep(1000);
       }
     }
     catch(InterruptedException e)
     {
       System.out.println("my thread interrupted");
     }
     System.out.println("mythread run is over" );
   }
}

class RunnableExample
{
    public static void main(String args[])
    {
      Count cnt = new Count();
      try
      {
        while(cnt.mythread.isAlive())
        {
          System.out.println("Main thread will be alive till the child thread is live");
          Thread.sleep(1500);
        }
      }
      catch(InterruptedException e)
      {
        System.out.println("Main thread interrupted");
      }
      System.out.println("Main thread run is over" );
    }
}
```

```
My thread created Thread[My runnable thread,5,main]
Main thread will be alive till the child thread is live
Printing the count 0
Main thread will be alive till the child thread is live
Printing the count 1
Main thread will be alive till the child thread is live
Printing the count 2
Main thread will be alive till the child thread is live
Printing the count 3
Main thread will be alive till the child thread is live
Printing the count 4
Main thread will be alive till the child thread is live
My thread run is over
Main thread run is over
Press any key to continue...
```

5. [4 Marks]

a) What will be the output of the program? [1 Mark]

```
import java.lang.*;


public class Q126 implements java.lang.Runnable
{
    private int x;
    private int y;

    public static void main(String [] args)
    {
        Q126 that = new Q126();
        Thread t1 = new Thread (that);
        Thread t2 = new Thread (that);
        t1.setName("Thread t1"); t2.setName("Thread t2");
        t1.start(); t2.start();
    }
    public synchronized void run( ) /* Line 18 */
    {
        for (;;) /* Line 20 */
        {
            x++;
            y++;
            System.out.println(Thread.currentThread().getName() +
                ": " + "x = " + x + " y = " + y);

            try
            {
                // Sleep for five thousand milliseconds (5 secs),
                // to simulate work being done
                Thread.sleep(5000);
            }
            catch (InterruptedException ie) {}

        }
    }
}
```

```
Thread t1: x = 30 y = 30
Thread t1: x = 31 y = 31
Thread t1: x = 32 y = 32
Thread t1: x = 33 y = 33
Thread t1: x = 34 y = 34
Thread t1: x = 35 y = 35
Thread t1: x = 36 y = 36
Thread t1: x = 37 y = 37
Thread t1: x = 38 y = 38
Thread t1: x = 39 y = 39
Thread t1: x = 40 y = 40
Thread t1: x = 41 y = 41
Thread t1: x = 42 y = 42
Thread t1: x = 43 y = 43
Thread t1: x = 44 y = 44
Thread t1: x = 45 y = 45
Thread t1: x = 46 y = 46
Thread t1: x = 47 y = 47
Thread t1: x = 48 y = 48
Thread t1: x = 49 y = 49
Thread t1: x = 50 y = 50
Thread t1: x = 51 y = 51
Thread t1: x = 52 y = 52
Thread t1: x = 53 y = 53
Thread t1: x = 54 y = 54
```

It will go to infinity.

b) What will be the output if we replace line 20 by: **[1 Mark]**

```
for (int i=0;i<5;i++) /* Line 20 */
```

```
Thread t1: x = 1 y = 1
Thread t1: x = 2 y = 2
Thread t1: x = 3 y = 3
Thread t1: x = 4 y = 4
Thread t1: x = 5 y = 5
Thread t2: x = 6 y = 6
Thread t2: x = 7 y = 7
Thread t2: x = 8 y = 8
Thread t2: x = 9 y = 9
Thread t2: x = 10 y = 10
Press any key to continue...
```

c) What will be the output if we remove the keyword *synchronized* in line 18? **[1 Mark]**

```
Thread t2: x = 4 y = 4
Thread t1: x = 4 y = 4
Thread t2: x = 5 y = 5
Thread t1: x = 5 y = 5
Thread t2: x = 6 y = 6
Thread t1: x = 6 y = 6
Thread t2: x = 7 y = 7
Thread t1: x = 7 y = 7
Thread t1: x = 8 y = 8
Thread t2: x = 8 y = 8
Thread t1: x = 9 y = 9
Thread t2: x = 9 y = 9
Thread t1: x = 10 y = 11
Thread t2: x = 10 y = 11
Thread t2: x = 11 y = 13
Thread t1: x = 11 y = 13
Thread t1: x = 12 y = 15
Thread t2: x = 12 y = 15
```

d) What will be the output if we remove the keyword *synchronized* in line 18 and replace line 20 as in part b above? **[1 Mark]**

```
Thread t2: x = 2 y = 2
Thread t1: x = 2 y = 2
Thread t2: x = 4 y = 4
Thread t1: x = 4 y = 4
Thread t1: x = 5 y = 5
Thread t2: x = 5 y = 5
Thread t1: x = 6 y = 7
Thread t2: x = 6 y = 7
Thread t2: x = 8 y = 9
Thread t1: x = 8 y = 9
Press any key to continue...
```

QUESTION 3: [5 Marks]

What will be the output of the programs?

a) **[1 Mark]**

```
class Table{

  void printTable(int n){
     synchronized(this){//synchronized block
       for(int i=1;i<=5;i--){
        System.out.println(n*i);
        try{
         Thread.sleep(400);
        }catch(Exception e){System.out.println(e);}
       }
     }
  }//end of the method
  }

public class TestSynchronizedBlock2{
  public static void main(String args[]){
  final Table obj = new Table();//only one object

  Thread t1=new Thread(){
  public void run(){
  obj.printTable(5);
  }
  };
  Thread t2=new Thread(){
  public void run(){
  obj.printTable(100);
  }
  };

  t1.start();
  t2.start();
  }
  }
```

```
5
10
15
20
100
200
300
400
Press any key to continue...
```

b) [1 Mark]

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<=5;i--){
    try{Thread.sleep(500);}
    catch(InterruptedException e)
        {System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();
  t2.start();
 }
}
```

```
1
1
2
2
3
3
4
4
5
5
Press any key to continue...
```

c) [1 Mark]

```
class TestCallRun2 extends Thread{
    public void run(){
        for(int i=1;i<=5;i--){
            try{
                Thread.sleep(500);
                }
            catch(InterruptedException e){
                System.out.println(e);
                }
            System.out.println(i);
          }
    }
    public static void main(String args[]){
     TestCallRun2 t1=new TestCallRun2();
     TestCallRun2 t2=new TestCallRun2();

     t1.run();
     t2.run();
    }
}
```

```
1
2
3
4
5
1
2
3
4
5
Press any key to continue...
```

d) [1 Mark]

```
class TestJoinMethod1 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try{
  t1.join();
 }catch(Exception e){System.out.println(e);}

 t2.start();
 t3.start();
 }
}
```

```
1
2
3
4
5
1
1
2
2
3
3
4
4
5
5
Press any key to continue...
```

e) [1 Mark]

```
class TestJoinMethod3 extends Thread{
  public void run(){
   System.out.println("running...");
  }
 public static void main(String args[]){
  TestJoinMethod3 t1=new TestJoinMethod3();
  TestJoinMethod3 t2=new TestJoinMethod3();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());
  System.out.println("id of t1:"+t1.getId());

  t1.start();
  t2.start();

  t1.setName("Sonoo Jaiswal");
  System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

```
Name of t1:Thread-0
Name of t2:Thread-1
id of t1:10
After changing name of t1:Sonoo Jaiswal
running.....
running.....
Press any key to continue...
```

g. (T) Constructing a server socket is as simple as specifying the port you want to listen on, like this:

```
try {
        ServerSocket ss = new ServerSocket(80);
}
catch (IOException e) {
        System.err.println(e);
}
```

h. (T) When a `ServerSocket` object is created, it attempts to *bind* to the port on the local host given by the port argument. If another server socket is already listening to the port, then it waits until the connection closed and tries to bind again.

2.5

QUESTION 2: [8 Marks]

1. Explain what is meant by: thread, daemon thread, synchronization and deadlock **[2 Marks]**

Thread: a sequence of responses to an initial message posting.

Daemon thread: those thread which runs in background and mostly created by JVM for performing background task like Garbage collection and other housekeeping tasks.

Synchronization: is the concurrent execution of two or more threads that share critical resources.

Deadlock: a situation where two or more threads are blocked forever, waiting for each other.

2. Consider the following code:

```
class X implements Runnable
{
    public static void main(String args[])
    {
       /* Missing code? */
    }
    public void run() ()
}
```

How can you start a thread? **[1 Mark]**

```
X run = new X();
Thread t = new Thread(run);
t.start();
```

3. Which will contain the body of the Thread? **[1/2 Mark]**

a. run() b. start() c. stop() d. main() e. body()

4. Which two of the following methods are defined in class Thread? **[1/2 Mark]**

a. start() b. wait() c. notify() d. run() e. terminate()

Q3. What will be the output of the program? [2 Marks]

```
// TestThreads.java
class MyThread extends Thread
{
    MyThread()
    {
        System.out.print(" MyThread");
    }
    public void run()
    {
        System.out.println(" foo");

    }
    public void run(String s)
    {
        System.out.println(" baz");
    }
}
public class TestThreads
{
    public static void main (String [] args)
    {
        Thread t = new MyThread();
        t.start();
    }
}
```

// wont run because no Strings were given

MyThread foo